HONOURS RESEARCH PROJECT REPORT

CLASS CONSTRUCTION AND STUDENT ALLOCATION

by

Dennis P. Chang

Department of Computer Science

University of Canterbury

TABLE OF CONTENTS

****

1. INTRODUCTION

1.1 BACKGROUND

Each year in every secondary school, days or even weeks are spent in constructing a school timetable. At present, this is done by senior staff (whose time could be better spent on something else), since timetable construction requires knowledge, and authority to make decisions, that only they have.

1.2 NOMENCLATURE

In timetabler's parlance, 'block' refers either to a set of classes, all of which always occur at the same time, or to a set of periods during which such a block of classes occurs. For example,

If Mr. Smith is taking 5B2 English in periods 2, 7, 14 and 20 then if Mr. Brown is taking 7B1 Mathematics in period 2 then he will also be taking that same class in period 7, 14 and 20.

A timetable which exhibits such feature is said to be 'blocked'. The senior timetables of schools are frequently blocked.

1.3 PROBLEM TO BE INVESTIGATED

The timetabler's problems are

1. which periods should occur in which block
2. which subjects should occur in which block
3. to which classes should each student be allocated

The aim of this research project was to investigate the problem of class construction and student allocation, and

derive techniques which could be used to solve the problem. In order to be of practical value to schools wishing to make use of these techniques, they must be able to be implemented on a mini / micro-computer and take no more than 17 hours of computing time, so that it can be run overnight.

2. ENVIRONMENT AND A BRIEF ANALYSIS OF THE PROBLEM

2.1 REQUIREMENTS

The problem of class construction and student allocation is to be investigated with a view to constructing a program that will, at least in part, meet the following requirements:-

1. as many student choices as possible are satisfied.
2. constraints imposed by availability of specialist rooms etc are not exceeded.
3. the number of classes created is, as far as possible minimised.
4. subject to one to three above, classes are reasonably homogeneous with respect to ability level and/or background.

2.2 ENVIRONMENT

2.2.1 CONSTRAINTS

The constraints imposed on the problem are

1. Basic constraints

For each subject the following will be limited:

(i) number of students allowed per class

(ii) number of classes allowed in each block

(iii) total number of classes allowed

2. Group Constraints

Because of limited resources of specialist rooms and number of teachers available, a limit may also be imposed on the total number of classes for a group of subjects in any block, e.g. the total number of science classes may be limited as well as the number of classes in particular sciences.

A subject may belong to as many group constraints as desired.

The limit on the number of classes allowed for each subject can be viewed as a group constraint imposed on all blocks for a subject.

2.2.2 INPUT DATA

The number of blocks, the subjects applied by each student, the basic constraints and the group constraints, if any, are to be supplied by the user.

It will be assumed that the number of subjects to be taken by each student is the same as the number of blocks. This may require the invention of filler subjects, e.g. study.

2.3 ANALYSIS

The problem grows exponentially with the number of students.

Assuming 6 blocks, the first subject can be allocated to any of the 6 blocks, the second subject can be allocated to any of the remaining 5 blocks and so on, a total of 6! = 720 possible ways to allocate each student. So there will be 720^n possible ways to allocate n students. Because constraints are imposed on the problem the complexity may be somewhat reduced.

3. RESEARCH DIRECTIONS INVESTIGATED

3.1 DIVISIONS OF THE PROBLEM

The problem of class construction and student allocation can be viewed as two separate problems: (i) class construction, (ii) student allocation. The constraints for the class construction problem are the basic and group constraints specified in section 2.1.1 while any feasible solution obtained is used as the basis for the student allocation problem. But a feasible solution obtained for the class construction problem, when used as basis for the student allocation problem, may preclude the existence of a feasible solution to the student allocation problem (see fig. 1). Thus, it is important to be able to find a feasible solution to the class construction problem quickly in which case optimality can be ignored.

Various methods have been investigated; section 3.2 contains discussions on methods that show some promise.

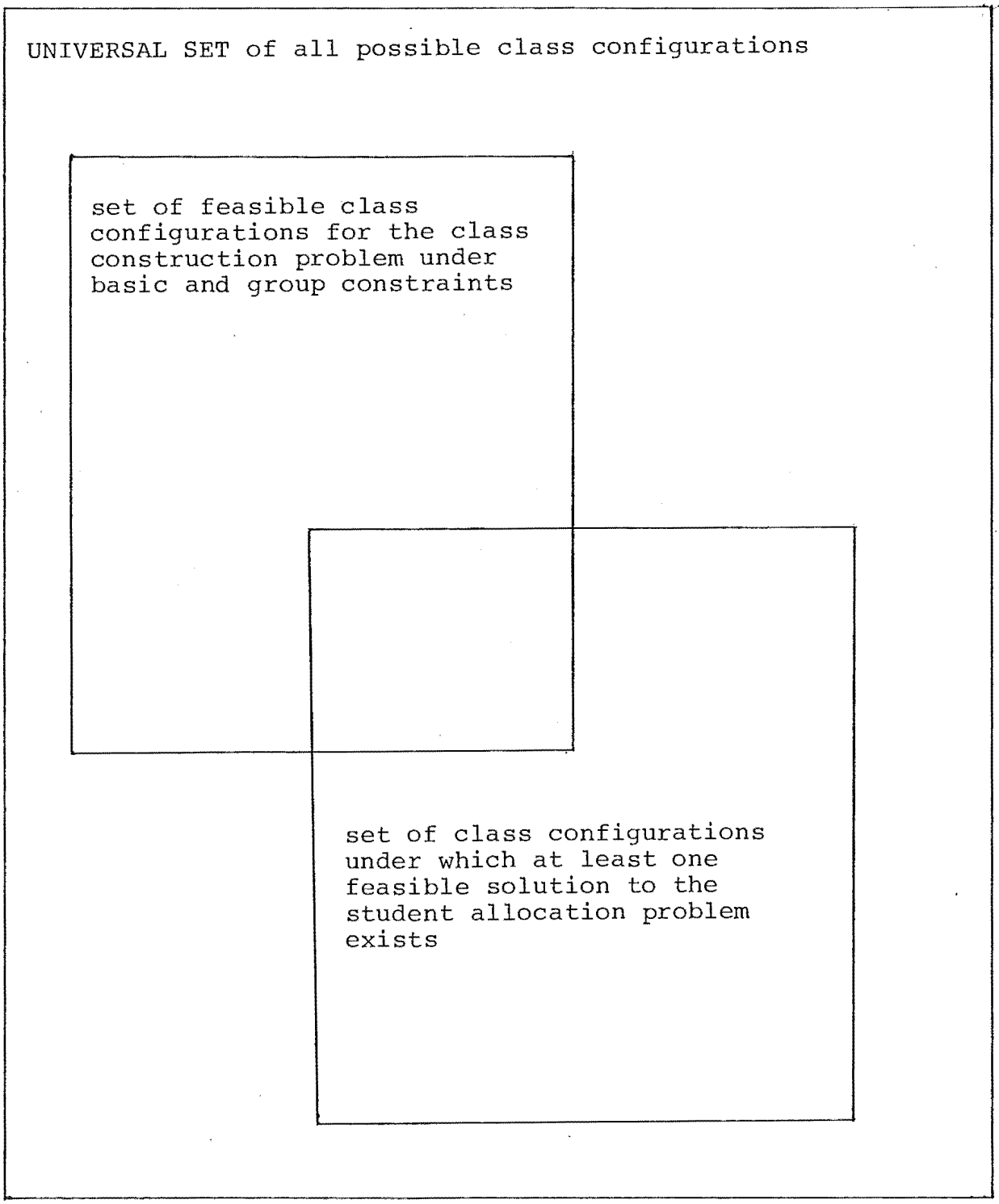

UNIVERSAL SET of all possible class configurations
set of feasible class configurations for the class construction problem under basic and group constraints
set of class configurations under which at least one feasible solution to the student allocation problem exists

Figure 1

3.2 SOLUTION METHODS INVESTIGATED

Linear Programming

(reference 1 - 3)

The problem of class construction can be formulated as a linear programming problem (Appendix B). It has been shown by Hacijan (Ref. 3) that a polynomial algorithm exists for linear programming, but since the solution obtained using linear programming techniques is not necessarily integral and we cannot have a non-integral number of classes, linear programming is of little value to us.

Integer Programming

(reference 1, 4 - 7)

Using the same formulation as linear programming (Appendix B), with the additional condition that all variables are to take integral values, we can solve the class construction problem using integer programming. A solution method for the student allocation problem can then be based on the solution obtained for the class construction problem using integer programming.

Performance figures contained in Geoffrian and Marsten (Ref. 6) for various integer programming algorithms gave an indication of the amount of computing time required for our problem. For a class construction problem with 50 subjects and 6 blocks would take 20 minutes computing time on an IBM 360/85 to obtain optimal solution using Davis, Kendrick and Weitzman's algorithm (Ref. 6). Integer programming is

a good method to use to solve the class construction problem.

Trauth and Woolsey (Ref. 7) contains results of testing 29 integer linear programming problems using 4 different computer codes.

In theory, every integer linear programming problem can be solved. That is, there are procedures which can be used that will lead to a solution of the problem with only a finite number of computations involved. In practice, however, several factors limit the use of these procedures. One factor is the time invloved in obtaining a solution. Also, it is necessary to consider the round-off errors and machine size in determining whether a problem can be solved. This is complicated further by the fact that problems which are extremely difficult or impossible to solve using a given integer linear programming code, may be much more easily solved using another code. In general, there is no way to predict such an occurrence. Woolsey (Ref. 8) gave potential users of integer programming some disconcerting advice: "to reconsider carefully before using any integer programming code". Four real examples are cited.

We can use integer programming to solve the class construction problem, and if the problem is feasible, obtain a class configuration with the minimum number of classes created.

Zero-One Linear Programming

(Reference 1, 4 - 8)

A formulation in which variables can only take the value of zero (no) or one (yes) is called a zero-one formulation, and, if the constraints are linear, then we have a zero-one

linear formulation.

Zero-One linear programming can be used to formulate the student allocation problem (Appendix C), provided a solution to the class construction problem has been found using some other methods (e.g. integer programming, heuristic methods).

Because of excessive computing time and storage requirements it is only practical to use zero-one linear programming to solve small to medium sized problems (e.g. a few hundred zero-one variables). Some heuristic zero-one programming methods seem more attractive.

Heuristic Zero-One Programming

(Reference 9 - 11)

Zanakis (Ref. 9) examines the performance of 3 heuristic methods when applied to zero-one problems. The general result is encouraging: the best results obtained with 1000 zero-one variables is 360 seconds CPU time on an IBM 360/75; the results obtained all lie within 5% of the optimal solution while taking only a small fraction of the time required by the IBM/MPSX code.

Because of the size of our problem, even this favourable result does not mean it can be applied to our problem. Using the formulae supplied by Zanakis, we find that the estimated computing time required for a problem with 36,000 0-1 variables (1000 students, 30 subjects and 6 blocks) is 18 days. This is too long for our purpose.

General Heuristic Methods

No tailor-made heuristic method has been found for the class construction and student allocation problem.

3.3 REVIEW AND EVALUATION OF METHODS

In Section 3.1 we examined methods which show some promise towards solving our problem. Linear programming is of little use because the solution is not necessarily integral, however it can be used to find a starting point, when using integer programming or heuristic methods, to reduce the computational time required. Integer programming can be used to obtain a solution to the class construction problem. Zero-one linear programming has no practical value because of the size of our problem. Heuristic zero-one programming shows slightly more promise but is still impractical because of excessive computing time and storage required.

Since what constitutes optimality is often unclear, only a feasible solution is required. Even though this will reduce the computing time in some cases, the large size of our problem requires more drastic reduction in computing time. A specially designed heuristic method utilising the special properties of the problem is desirable.

4. DESCRIPTION OF HEURISTIC METHOD DEVISED

4.1 OVERVIEW

To utilize the special properties (basic and group constraints, and blocked structure) of the problem, an attempt has been made to design and implement a heuristic method for the class construction and student allocation problem.

The method consists of three phases. Together, phases one and two attempt to obtain a feasible solution to the class construction problem, all constraints being satisfied.

Phase 3 attempts to allocate students to the classes created by phase 2. It is recognised that a complete solution is most unlikely to be found because of the large number of possible ways to allocate students and impracticality of determining whether a feasible solution exists until one has been found.

Both phases 1 and 2 base their method on path finding, which will be described in detail in section 4.2. Phase 3 only uses path finding if a difficult case arises. In what follows distinction is made between "places created", which are vacancies that we hope will be taken up by students, and "students allocated" which are individual students actually allocated classes for the subjects they apply.

4.2 FINDING FLOW-PATHS IN A DYNAMICALLY CHANGING DIRECTED GRAPH

4.2.1 WHAT IS A PATH - AN EXAMPLE

Consider the situation of two warehouses (W1 and W2) which can store 2 different types of commodities (C1 and C2), each type of commodity takes the same storage space. Each warehouse can store a maximum of 24 units of commodities, and there is also a limit on the number of each type of commodity that can be stored, the two limits must add up to no less than 24. The total number of commodities to be stored in both warehouses is the same as the total capacity of the two warehouses.

Given the following information, how do we assign the commodities using some systematic procedure?

	Total	W1 Limit	W2 Limit
C1	22	16	12
C2	26	18	16

(i) We start off by assigning as much C1 to W1 as we can, which is 16, there are now 6 C1 left. We now assign as much C1 to W2 as we can, which is the minimum of what is left of C1 and what W2 can take under the C1 storage limit and total warehouse capacity. This gives us 6. Now we try to assign C2 to W1. Since W1 has already got 16 units of C1, it can only take 8 units of C2. This will leave 18 units of C2 left to be assigned to W2. But even though W2 can still take (24-6)=18 units of commodity, it can only take a maximum of 16 units of C2. After assigning 16 units of C2 to W2 we are left with 2 units of C2 and W2 requiring 2 units of commodities.

	Yet to be Assigned	W1	W2
C1	0 (22)	16 (16)	6 (12)
C2	2 (26)	8 (18)	16 (16)
		24	22

We quickly realise that we need just shift 2 units of C1 from W1 to W2 and assign 2 more units of C2 to W1. But how can we do this systematically?

We have 2 units of C2 yet to be assigned. Is there any warehouse that can still take some more C2? Yes, W1. But since W1 has already got 24 units of commodities, we must shift some C1 out of W1 if any more C2 is to be assigned to W1. Now, which warehouse can still take some more C1?

Since W2 can still take some more commodities (not full) we need not shift out anything. So we will assign 2 more units of C2 to W1, shift 2 units of C1 to W2 and all is well. Thus, a path is found (un-closed).

	W1	W2
C1 0(22)	14 (16)	8 (12)
C2 0(26)	10 (18)	16 (16)

(ii) If we need to reduce C1 held at W2, we cannot because there is no remaining storage capacity left for other types of commodities (in this case C2). In this case, we say that closed path can be found for C1 originating from W2. But if we want to reduce stock for C2 at W2, then a closed path can be found: reduce C2 at W2, increase C2 at W1, reduce C1 at W1 and increase C1 at W2. The maximum flow that can be made through this path is the minimum that can be increased or decreased, which is 4 since C1 can only be increased by 4 in W2.

	W1	W2
C1	10 (16)	12 (12)
C2	14 (18)	12 (16)

4.2.2 PATH FINDING

The example given in 4.2.1 is a scaled down version of what is done in phase 1 and 2 of the heuristic method. Types of commodities correspond to subjects, warehouses to blocks, unassigned commodities in a commodity type to unsatisfied demand for a subject, total capacity for a warehouse to the total number of students (all of whom must be having a class in every block) and the limit of each commodity type in warehouses corresponds to the number of students allowed in each class for a subject multiplied by the number of classes allowed for each block (subject-block constraint).

What we must also take into account when deriving a feasible solution for the class construction problem is (i) the limit on the number of classes allowed for each subject (an increase/decrease in the number of classes created for a block will decrease/increase the number of classes that still can be created in other blocks for the same subject).
(ii) if group constraints are also imposed on the subject in the block in which the number of classes has been changed, this change will have an effect on all other subjects restricted by the same group constraint.
(iii) the different class sizes allowed for each subject.

All of these do not change the method used to find paths but they do call for a more sophisticated method of altering the capacities of each subject in each block; a more complicated algorithm is also used to select the best path and the maximum flow.

We can picture this as a graph using nodes for blocks; each code having as many layers as there are subjects. For each layer there is a directed edge from the node (if it has places created in it) to nodes with capacity to take more places in that subject. A node is often both an originating and destination node in a particular layer. The originating node and destination node of an edge must be different.

In practice, because of their highly dynamic nature, the edges are not stored. Instead, six tables are used to keep track of the complete status of the class construction and student allocation process. Briefly, the information available for each subject in each block is:

1. number of students allocated (only used in Phase 3)
2. number of student places created
3. number of classes created
4. number of student places that can still be created without creating a new class
5. number of student places that must be removed to abolish one class
6. capacity, i.e. number of classes that can still be created or number of classes that must be reduced. This figure takes into account all basic and group constraints.

Note: A new class is created only if the previous class created in that subject is full.

The number of classes created (Table 3) includes the last class created which may not be full. The number of student places that must be removed to abolish one class (Table 5) is the same as the number of student places created in the last class created. If more than one class needs to be abolished, a number of student places equal to the number of students allowed per class for the subject must be removed for every additional class to be abolished. The number of student places that can still be created without creating a new class (Table 4) is the number of student places we can still create in the last class created before that class is full, (this is zero if the class is full).

Tables 3, 4 and 5 can be constructed from the number of students allowed per class and Table 2 (which contains the number student places created for each subject in each block, which is also the number of student places that can be removed). In phase 3, the difference of Table 2 and 1 must be taken because in path finding we can only shift student places not already allocated.

The capacity table (Table 6) indicates the number of classes that can still be created for each subject in the block (while still satisfying all constraints) or the number of classes of each subject in the block that have been created in excess of the block capacity. A change in the number of classes created in a block for a subject will alter the capacities of all other blocks in the same subject and of

all other subjects in the same block and constraint groups. So, if a block in a subject has not yet satisfied all the constraints, the number of classes created in other blocks in the subject and subjects in the same block and group constraints may also have to be reduced before the constraints can be satisfied. Similarly, the number of classes that can still be created in a block may be reduced because new classes have been created in other blocks in the same subject or in the same block for different subjects which are under the same group constraints.

From Tables 4, 5, 6 and the number of students allowed per class we can calculate the capacity of a block in a subject in units of numbers of student places created. This is needed by the path-finding procedure to compute the maximum number of student places we can flow through a particular path.

A path is found by matching the outlets from one block with inlets of blocks not already in the path and then looking for outlets in the latter. This process continues until either an unfilled block (for Phase 1) or a closed path is found (Phase 2 and 3). Except for the starting block (which will have only one outlet) all other blocks on the path could have more than one outlet. A widthfirst search is conducted after which several paths of the same length may have been found. The maximum length possible is the same as the total number of blocks.

This process guarantees that if a path exists it will always be found.

4.3 PHASE ONE - INITIAL PREPARATION

Phase 1 attempts to fill up all blocks and satisfy demand for all subjects. Only the limit on the number of classes for each subject in each block is observed. (See part (i) of example in section 4.2.1)

The larger the number of group constraints imposed on a subject in a block, the higher the probability that the capacity of other subjects in the same group constraints and other blocks in the same subject will be reduced if classes are created for the subject in the block. In the worst case, no capacity will remain for any other blocks in the same subject or other subjects in the same group constraints. To minimize the impact, a table containing the number of group constraints on each subject for each block is setup. Student places are created starting with subject-blocks with the minimum number of group constraints and proceeding to subject-blocks with the highest number of group constraints. This is done to reduce the amount of reassigning of student places required in Phase 2 if not all constraints are satisfied after Phase 1.

The following procedure is performed for subject-blocks on which the same number of group constraints are imposed.

The subjects are processed sequentially while the blocks for each subject are processed serially-randomly; that is, the starting block is picked randomly and the blocks are treated as a circularly linked list. This is done to minimize clustering of classes in blocks. For blocks with the number

of group constraints we are dealing with, we assign as much of the remaining demand for the subject as possible without over-filling the block.

After this, the tables mentioned in Section 4.2 are made. If there is any unsatisfied demand we will search for paths from each of the subjects with remaining demand to unfilled blocks. A path table will be created if paths exist. If there is a choice of path, then the path with the largest flow is selected (to minimize the number of times the path table has to be constructed). Paths containing subject-blocks with an excess of classes will always be selected before paths which do not contain such subject-blocks.

Phase 2 will commence only if Phase 1 is successful.

4.4 PHASE TWO - CLASS CONSTRUCTION

In this phase, the number of classes in subject-blocks with an excess of classes is reduced. Since over-creation of classes can be the result of over-creation of classes in the subject and/or over-creation of classes in a constraint group to which the subject-block belongs, we may find that in some cases classes of other blocks or subjects will also need to be reduced for the constraints to be satisfied. Similarly, a reduction in the number of classes created in one block may make it unnecessary to reduce the number of classes created in other blocks or subjects. The Path-finding technique is used to resolve these problems.

Because of the reasons stated above, we can say that no feasible solution exists for the class construction process only if there are still subject-blocks left with over-created classes after we have processed all the subject-blocks with over-created classes, otherwise the class configuration obtained is a feasible solution to the class construction problem.

4.5 PHASE THREE - STUDENT ALLOCATION

Given a class configuration supplied by phase 2, phase 3 will attempt to allocate students using the following techniques.

4.5.1 ALLOCATE STUDENTS IN ORDER OF SUBJECTS

Suppose 6th form Economics is taught only in Block B in a timetable with only 3 blocks, then students taking 6th form Economics must have Economics in Block B. Consequently the possible choices of blocks for other subjects are limited. For example, if English is available only at Blocks A and B then students taking Economics all have to take English in Block A. We need to allocate these students first to avoid other students taking up the spaces in Block A English required by those taking 6th form Economics.

In general, students taking subjects which have fewer choices of blocks are allocated first.

4.5.2 BASIC ALLOCATION TECHNIQUE

For each student to be allocated, the subject with fewest choices of blocks, and blocks with fewer choices of subjects are allocated first. For example,

Blocks / Subjects	B1	B2	B3	B4
S1		1	1	
S2	1	1		1
S3		1	1	
S4			1	1

Since only S2 can be allocated to B1 we will allocate S2 to B1 first (note: A for Allocated).

	B1	B2	B3	B4
S1		1	1	
S2	A			
S3		1	1	
S4			1	1

Since S2 has been allocated to B1 it cannot also be allocated to other blocks.

Now only S4 can be allocated to B4.

	B1	B2	B3	B4
S1		1	1	
S2	A			
S3		1	1	
S4				A

At each point, we allocate the subject with the fewest available blocks. If there is more than one, one subject is picked at random then a block with classes created is picked at random. The process continues until we have allocated all subjects that can be allocated.

4.5.3 REALLOCATE STUDENTS ALREADY ALLOCATED

If we cannot allocate a student using the basic technique described in Section 4.5.2 then students already allocated are searched to see if any one of them can be reallocated so as to free a place for the unallocated student.

In the previous example, say John takes 6th form Economics, English and German. We must allocate Economics to Block B and English to Block A. Suppose we find we are unable to allocate German to Block C. This maybe because there never was a German class created in Block C, in which case there is nothing we can do at this stage. On the other hand, if there are student places created for German in Block C but they have all been allocated, clearly, there must be student places available in some other block, otherwise total demand will exceed the total number of student places created which cannot be the case if a feasible class configuration has been found in Phase 2. We now look for a student who takes German in Block C and is able to take German in some other block and take some other subject in Block C. If such a student is found, the subjects are exchanged and a vacancy will be created in Block C for German which will be allocated to John.

4.5.4 REDISTRIBUTE STUDENT PLACES CREATED

If a student cannot be allocated using the reallocation scheme, we will attempt to redistribute the created student places using path-finding techniques. That is, to modify, within the constraints, the class configuration provided by Phase 2 in the hope of creating an additional student place in the block to which a subject cannot be allocated.

If this also fails we will put the student on a "difficult list" waiting to be allocated with the last attempt at the end.

4.5.5 LAST TRY

Until now, we have only allocated students to student places created and ignored the fact that there may be classes in which more student places could be created because of remaining capacity. We have also assumed that all student places created will eventually be taken up. This is done to try to keep the final class configuration after student allocation as close as possible to the class configuration provided by Phase 2. Now, however, we check whether we would allocate any of the "difficult" students by using such remaining capacity. Note that this will not involve breaking any constraints - it is merely a matter of using the difference between the number of students we originally expected to allocate to a class, and the number of students we are permitted to allocate to that class.

5. EXPERIMENTAL RESULTS

The test data used are from the 1977 5th Form timetable from Linwood High School, Christchurch. The characteristics of each trial are shown by the following:

1. number of blocks
2. number of students
3. number of group constraints
4. supply ratio: average over subjects of the supply to demand ratio
5. block ratio: average over subjects of the number of blocks in which classes can be created to toal number of blocks
6. class ratio: average over subjects of the ratio of the sum of number of classes allowed in individual blocks to total number of classes allowed in individual blocks to total number of classes allowed in the subject.

The original student population is 369, a sample of 99 students was taken. There are six blocks, no group constraints and most students also take the subject "study". The results are shown in Table 1. The value in bracket for supply ratio is the supply ratio with out the subject "study". The trials were ran on a B6700 computer.

The computing time used in Phase 1 increased only slightly while total computing time increased more than linearly with the increase of problem size. The computing time increases with decrease in success rate (Trials 1, 2, 3 and 4). Low success rate seems to correlate with high class ratio (Trials 1, 2, and 3) while the decrease in supply ratio (Trials 1 and 4) does not seem to have any effect on

the success rate.

The reallocation technique is more useful with difficult problems (Trials 2, 3 and 5) while the redistribution technique has no success. No last attempt was needed for Trials 1 and 4 which had a 100% success rate.

In general, a tight constraint and good class configuration are two important elements in getting a high success rate in student allocation.

TABLE 1. RESULTS

TRIAL	No. of STUDENTS	No. of GROUP CONST-RAINTS	BLOCK RATIO	SUPPLY RATIO	CLASS RATIO	No. of STUDENTS ALLO-CATED	SUCCESS RATE (%)	COMPUTING TIME (SECONDS)			PATH FINDING CALLS IN PHASES 1 AND 2	No. of STUDENTS ALLOCATED USING		
								PHASE 1	PHASE 2	PHASE 3		REAL-LOCATION	REDIS-TRIBUTION	LAST ATTEMPT
1	99	0	0.433	1.308 (1.055)	1.075	99	100	1.3	0	2.8	5	1	0	0
2	99	0	0.450	1.308 (1.055)	1.250	96	97.0	1.6	0	5.9	28	2	0	3
3	99	0	0.692	1.308 (1.055)	2.260	92	92.9	1.9	0	8.4	53	6	0	3
4	100	0	0.433	1.526 (1.287)	1.075	100	100	1.4	0	2.7	4	1	0	0
5	365	0	0.433	1.308 (1.018)	1.075	357	97.8	1.6	0	15.2	24	6	0	2
6	369	0	0.433	1.294 (1.008)	1.075	362	98.1	1.7	0	15.9	28	1	0	3

6. PERFORMANCE OF THE METHOD

6.1 SOLUTION REQUIREMENTS CONSIDERATIONS

Let us examine the performance and capability of the heuristic method under the four requirements the project is trying to achieve.

1. As many student choices as possible are satisfied.

The result obtained for the largest problem tested has a success rate of over 97% using as input data a class configuration that is known to have a feasible solution. A scaled down version had a success rate of 100%. The results seem encouraging, but given a set of input data for which we do not know whether any solution exists, it would be difficult to judge the solution obtained. For example, a success rate of only 50% for a given problem could indicate unstableness of the heuristic method, with varying performance depending on the problem since perhaps a much higher success rate could be obtained using exhaustive enumeration; but if the best that the data would allow even to an omniscient timetabler is a 51% success rate, then the heuristic method has done very well indeed.

No attempt has been made in this project to investigate intensively the stableness of the heuristic method and the characteristics of the problem that are likely to give a better/poorer performance.

2. Constraints imposed by the availability of specialist rooms etc are not exceeded

These constraints are contained in basic and group constraints which will all have been satisfied when the class

construction process finds a feasible solution. If a feasible solution exists for the class construction problem it will always be found.

3. The number of classes created is, as far as possible minimised

This requirement is not achieved by the heuristic method. Integer programming could be used to solve the class construction problem while minimising the number of classes created.

However, because of numerous other reasons outside the environment of the problem, a class configuration with the number of classes minimised may not be acceptable to the timetabler who may wish to make various changes to it. The heuristic method can be used efficiently and interactively by the timetabler to make his/her changes. This is done by manipulating the constraints imposed on the problem: reducing or preventing any further creation of classes by tightening the existing constraints or imposing new ones, introducing new classes by relaxing or removing constraints and so on. In this way, the experience and knowledge of the timetabler can be used effectively in making a class configuration that is feasible under all constraints imposed on the problem, acceptable to the timetabler and optimised with regard to criteria within and without the environment of the class construction problem.

4. Classes are reasonably homogeneous with respect to ability level and/or background

No effort has been made in the heuristic method to achieve this requirement because no information is available in the input data. Section 8 contains suggestions on modifications and additions that can be made to the existing heuristic method to meet this requirement in part.

6.2 OTHER CONSIDERATIONS

For the heuristic method to be of any practical value, the following must be considered:

1. Ability to be implemented

The heuristic method represented has been successfully implemented on a Burroughs B6700. Section 7 contains suggestions and considerations for implementation on mini / micro-computers.

2. Time Requirement

When run on B6700, about 10 seconds of computing time was used to obtain a solution to a problem with 100 students, while 23 seconds of computing time were used to obtain a solution to a problem with 365 students, a growth rate of less than linear.

It is hard to estimate the amount of computing time required on mini / micro-computers since it will very much depend on the type of machine used, the amount of main store available and file access times. But judging from the processing used on B6700, the 17 hour guideline is unlikely to be exceeded.

7. IMPLEMENTATION CONSIDERATIONS AND SUGGESTIONS

7.1

The heuristic method described in Section 4 can be implemented in two separate stages: one for the class construction process and the other for the student allocation process.

The information that needs to be passed between stages is the table containing the number of student places created. The table containing the number of students already allocated will also need to be passed from stage two to stage three. All other tables can be constructed from these tables and the input data which will be stored as a file.

If main storage permits, the basic and group constraints should be in main store for all three stages. The routines to read files into main store and to set up initially all tables required can be discarded once they are used so they would not take up valuable space in main store when the computations start. These routines will be used by both stages.

Stage 1 consists of Phase 1 and Phase 2 of the heuristic method. Its function is to derive a feasible class configuration if one exists or to inform the timetabler if a solution cannot be found. The procedures included are: the path-finding procedure, and control procedures for Phases 1 and 2.

The student allocation process is performed in the second stage. Because both path-finding procedures and Phase 3 procedures are large they may not be able to be implemented together in one program. In this case a variation can be introduced to allow the student allocation process to be

implemented in two separate programs. Program 1 will allocate students using the basic allocation technique and the reallocation technique, while the redistribution technique and final allocation technique will be implemented in the second program.

This means that the redistribution technique will be used to allocate students not so far allocated only after all students have been processed by the basic and reallocation technique. If this also fails, a last attempt will be made to allocate those students not yet allocated using the technique described in Section 4.5.5.

7.2

If possible, the file containing the subjects applied for each student should be brought into main store to save file access time. The program implemented on the B6700 using the heuristic method described requires only one word for each subject applied and an additional word to say how many subjects have been allocated for each student. If byte addressing is possible on the machine used then only 7 bytes are required for each student.

8. EXTENSION AND FUTURE RESEARCH

Extensions that can be made to improve the performance of the heuristic method are discussed under each requirement.

Requirement 1

(i) An improvement that can be made to the student allocation technique is in the method used in the basic allocation technique (Section 4.5.2). Instead of allocating all subjects a student applied for we could allocate only those subjects where only one choice of block remains and allocate subjects to blocks to which only one subject can be allocated (see example in Section 4.5.2). In this way, only the critical subjects for each student are allocated while no allocations are made for subjects with more than one choice of block. This will give those students who can only take those subjects in certain blocks the first chance to be allocated.

Each time a reduction of available blocks occurs in a subject, students taking that subject should be scanned to see whether any further choices are forced.

This improvement will nearly double the record size for each student since we now must keep track of which subjects have been allocated and to which blocks they have been allocated.

(ii) At present, we have no means of deciding when a class configuration is better than others. It may be that a set of criteria can be devised to avoid class configurations that are likely to give a poor allocation success rate. We might try to devise an algorithm to calculate the likely

number of classes that should be allocated to each block for the set of students to be allocated. A pre-processor to order students taking into account not only subject ability and background but also the class configuration provided by the class construction process may produce useful improvements.

Requirement 2

This requirement has already been met completely.

Requirement 3

Some optimising code (e.g. existing integer programming code) can be included in Phase 2 to minimise the number of classes created.

Requirement 4

In addition to what has been mentioned in Section 6 about mixing the order of students in the input data we can have a separate process after student allocation process to reallocate students so that blocks in the same subject are reasonably homogeneous with respect to ability level and/or background.

This is done by supplying, in the input data, an estimated ability level for each student while the students are sorted according to background by a pre-processor to ensure clustering of students with similar backgrounds in the input data. The student adjusting process will use techniques parallel to those described in Section 4.5.3 to reallocate students between blocks if some blocks in a subject are shown to be inhomogeneous (based on some criterion set down by the time-tabler).

9. SUMMARY AND DISCUSSION

The class construction and student allocation problem is essentially two separate problems to be solved under the same constraints.

The class construction problem can be solved using integer programming to minimise the number of classes created or by using Phase 1 and Phase 2 of the described heuristic method to find a feasible solution. The heuristic method would be much faster (4 seconds vs 20 minutes) but human assistance is most likely to be required to obtain an acceptable class configuration. On the other hand, a class configuration with the minimum number of classes created obtained using integer programming may not be acceptable for reasons that lie outside the environment of the problem (e.g. psychological considerations, staff preferences or past experience leading to feelings that it wouldn't work). It is best to use integer programming to obtain an optimal solution (i.e. number of classes created is minimised) and if this is not acceptable then the heuristic method can be used to obtain an acceptable class configuration using the optimal solution as the starting configuration.

The only practical method for allocating students is the student allocation process (Phase 3) of the described heuristic method. It does not take too much computing time (23 seconds CPU on B6700 for 365 students) and the result obtained is quite acceptable (6 out of 365 students unable to be allocated, a success rate of over 98%.) These are the best results obtained and were produced from input data known to have a solution.

ACKNOWLEDGEMENT

Special acknowledgement is given to Mr. R. Harries, my Project Supervisor, for his assistance, guidance and infinite patience. The valuable opinions of Dr. D.G. McNickle, Dr. L.R. Foulds and Mr. F.T. Baird also acknowledged.

*****

APPENDIX A

SYMBOLS USED

1. Constants

B : number of blocks

S : number of subjects

P : number of students

Symbols used in basic constraints:

ℓ_i : number of students allowed per class for subject i

S_i : number of classes allowed for student i

b_{ij} : number of classes allowed for student i in block j

Symbols used in group constraints:

G_{ij} : number of classes allowed for the ith group constraint in block j

2. Variables

d_i : number of students wanting to do subject i

C_{ij} : number of classes constructed for subject i in block j

X_{ij} : number of students assigned to subject i in block j

A_{ij} : number of students allocated to subject i in block j

3. Sets

V_i : set of subjects taken by student i

Q_{ij} : set of subjects in the ith group constraint of block j

Note:

1. V_i and Q_{ij} are fixed for a given problem
2. All constants and variables are non-negative

APPENDIX B

Linear Programming Formulation for the Class Construction Problem

Basic Constraints

$$1 \leqslant i \leqslant S \quad , \quad 1 \leqslant j < B$$

$$C_{ij} \leqslant b_{ij}$$

$$\sum_{j=1}^{B} C_{ij} \leqslant S_i$$

$$\left(\sum_{j=1}^{B} C_{ij} \right) \ell_i \geqslant d_i$$

$$\sum_{i=1}^{S} (C_{ij} \ell_i) \geqslant P$$

Group Constraints

$$\Sigma C_{kj} \leqslant G_{ij} \quad \text{where} \quad K \in Q_{ij}$$

Objective Function:

To minimize the number of classes created

$$Z = \sum_{i=1}^{S} \sum_{j=1}^{B} c_{ij}$$

APPENDIX C

Zero-One Linear Formulation For Student Allocation Problem Given A Constructed Class Table

Let P_{ijk} be a 0-1 variable indicating whether student i is taking jth subject ($j \in V_i$) in block k.

[0 for no, 1 for yes.]

X_{jk} are the number of students assigned to subject i in block k. This is given as inputs to the problem.

$1 \leqslant i \leqslant P \qquad 1 \leqslant k \leqslant B$

1. Each student must be allocated one and only one subject in any one block.

$$\sum_{j \in V_i} P_{ijk} = 1$$

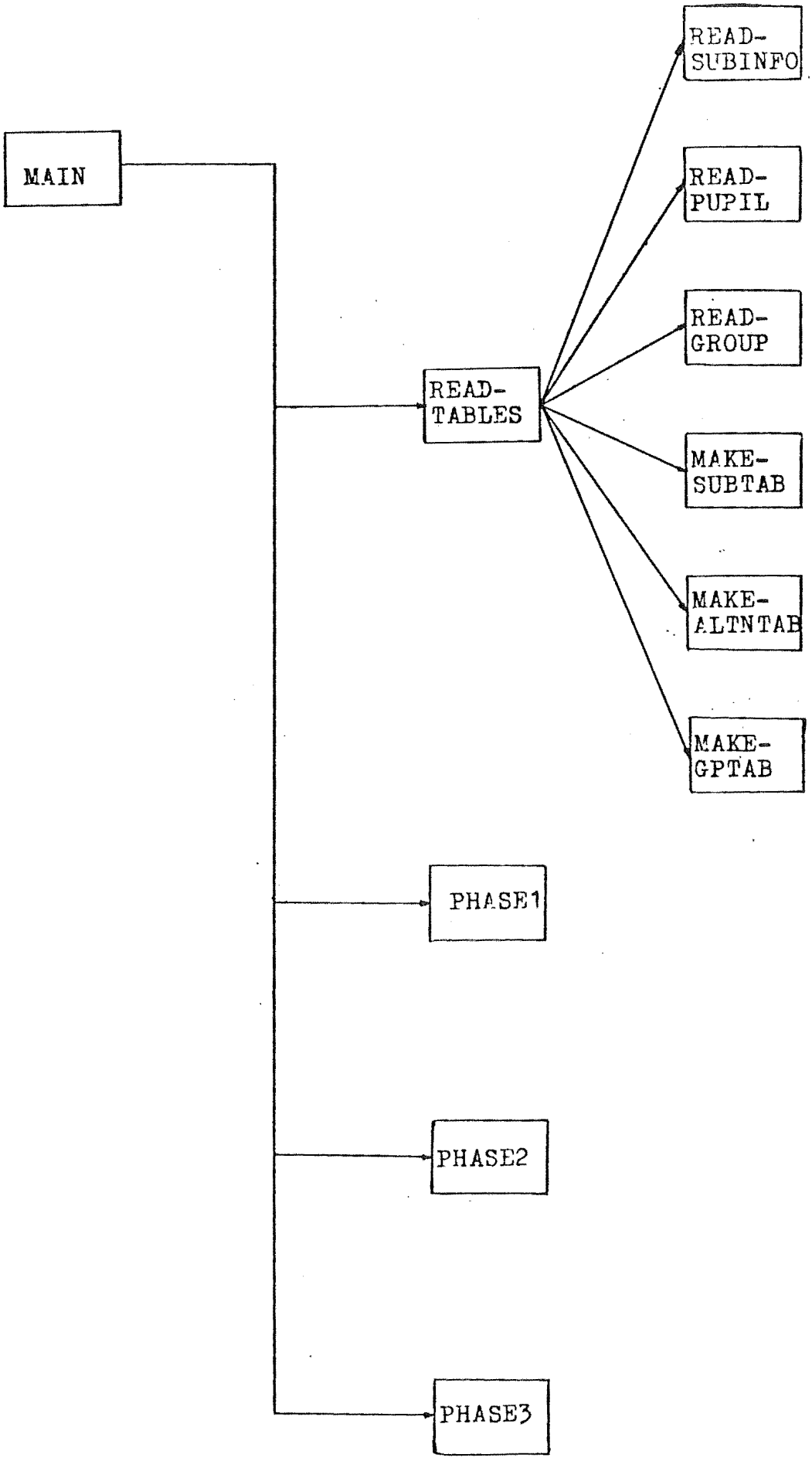
$$\sum_{k=1}^{B} P_{ijk} = 1 \quad , \quad j \in V_i$$

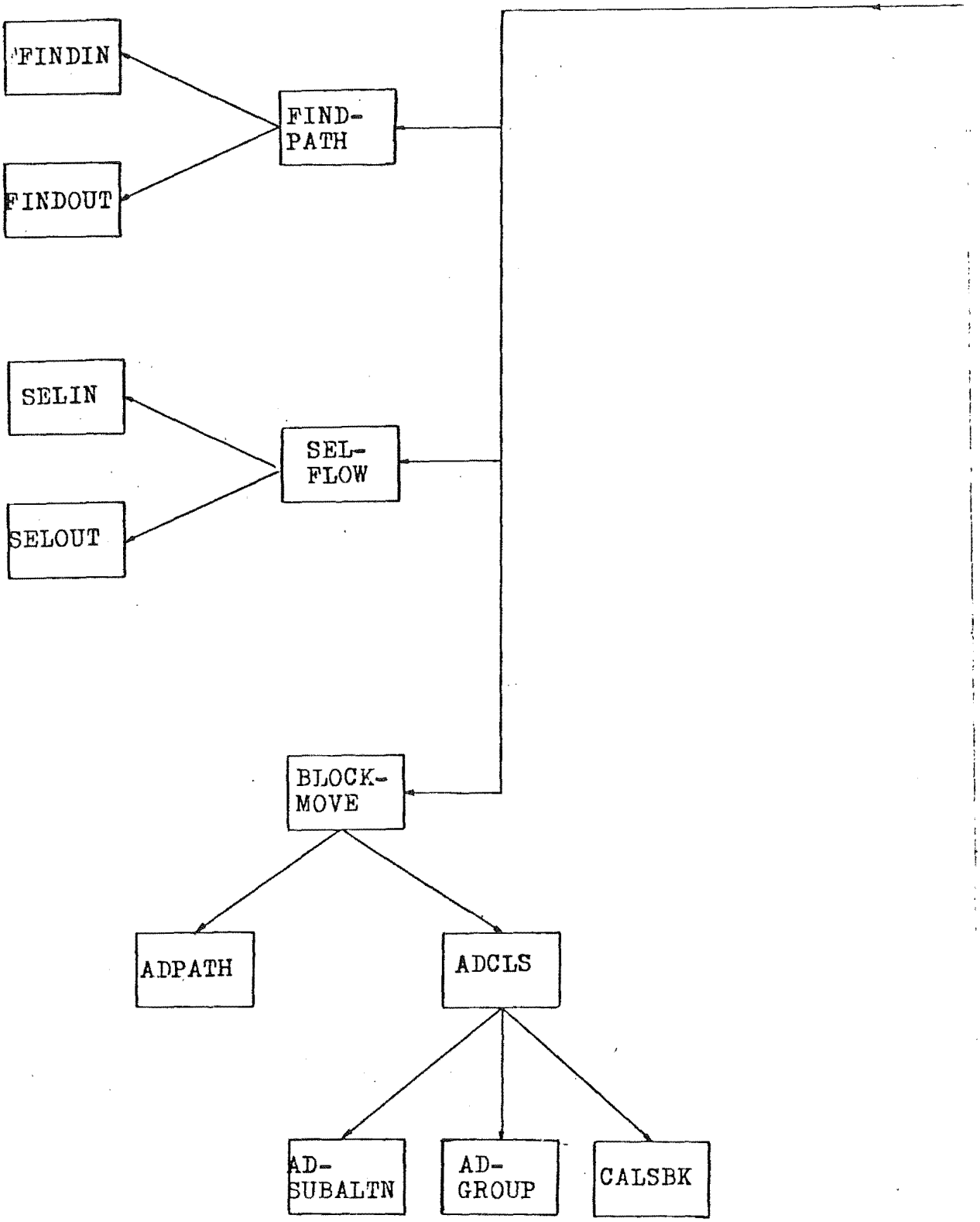
2. We must satisfy the class constraints imposed by the class table

$$\sum_{i=1}^{P} \sum_{k=1}^{B} P_{ijk} \leqslant X_{jk} \quad , \quad 1 \leqslant j \leqslant S$$

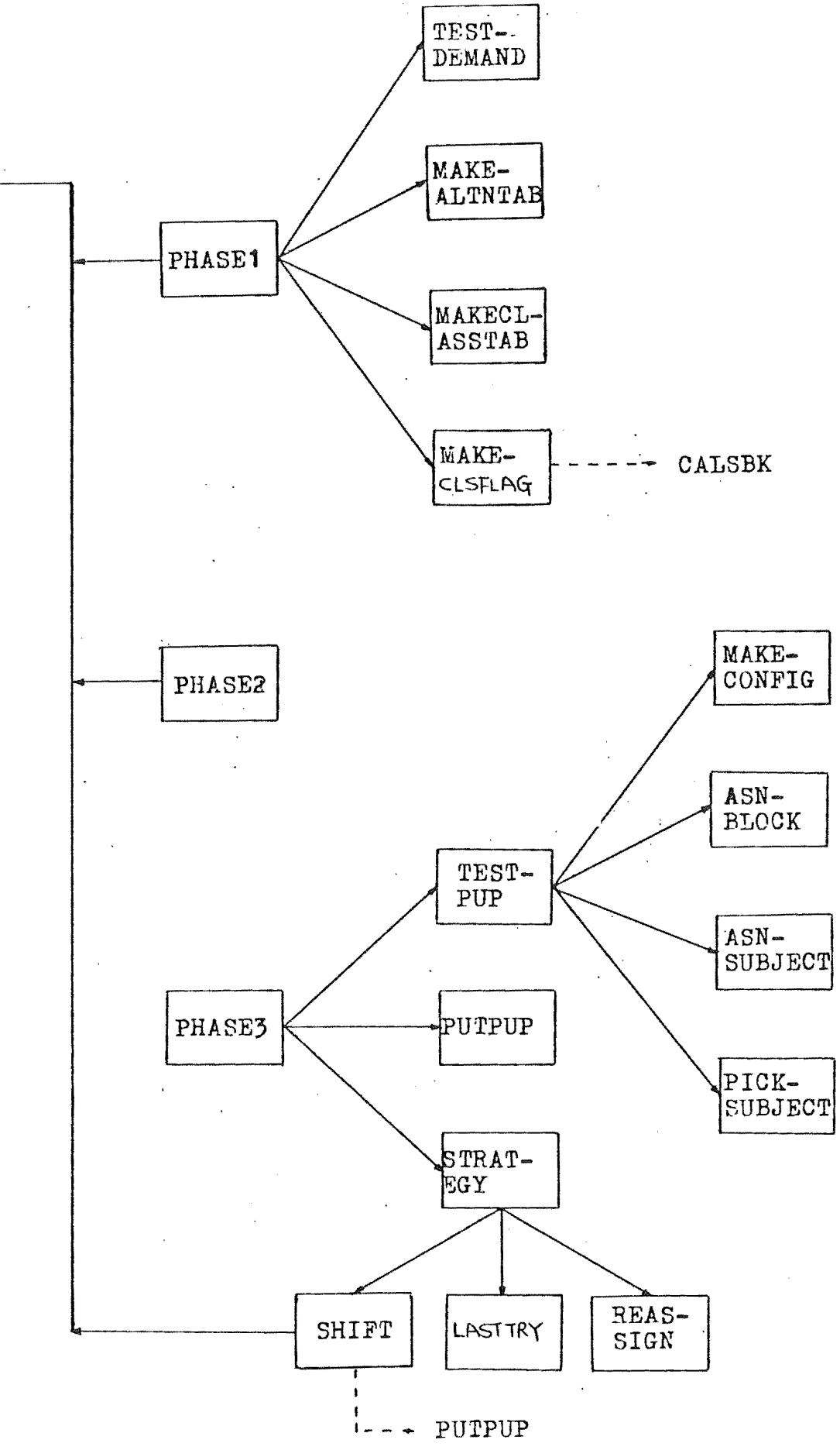
APPENDIX D

Procedure calling flow-chart for the program implemented on the B6700 using the heuristic method described in section 4.


MAIN
READ-TABLES
READ-SUBINFO
READ-PUPIL
READ-GROUP
MAKE-SUBTAB
MAKE-ALTNTAB
MAKE-GPTAB
PHASE1
PHASE2
PHASE3



PATH-FINDING PROCEDURES


TEST-DEMAND
MAKE-ALTNTAB
PHASE1
MAKECL-ASSTAB
MAKE-CLSFLAG
CALSBK
PHASE2
MAKE-CONFIG
ASN-BLOCK
TEST-PUP
ASN-SUBJECT
PHASE3
PUTPUP
PICK-SUBJECT
STRAT-EGY
SHIFT
LASTTRY
REAS-SIGN
PUTPUP

REFERENCES

General operational Research Text

1. Daellenback, H.G., and J.A. George
   Introduction to Operations Research Techniques
   Allyn and Bacon, 1978
   Good introductory book, give excellent treatment of a wide range of topics with examples.

2. Daellenback, H.G., and E.J. Bell
   User's Guide to Linear Programming
   Prentice-Hall, 1970
   Elementary; contain a listing of a linear program computer code with input/output and sensitivity analysis (LPGOGO)

Linear Programming and Simplex Method

See ref. 1 & 2

3. Hacijan, L.G.
   A Polynomial Algorithm in Linear Programming
   Soviet Math. Doklady, Vol. 20 (1979), No. 1

Integer Programming

4. Plane, D.R., and C. McMillan
   Discrete Optimization
   Prentice-Hall, 1971
   This book is a good place to start a more detailed study of integer programming without complicated mathematics. Contain computer codes for implicit enumeration of 0-1 proble, linear programming and a cutting plane algorithm.

5. Greengerg, N.
   Integer Programming
   Academic Press, 1971
   Intermediate to advance text. The theory is tempered with good applications and examples.

6. Geoffrion, A.M., and R.E. Marsten
   Integer Programming Algorithms: A Framework and State-of-the-art Survey
   Management Science, Vol. 18, No. 9, May 1972, pp. 465-491
   A good survey of integer techniques. Contain a list of extensive references.

7. Trauth, C.A. JR, and R.E. Woolsey
   Integer Linear Programming: A Study In Computational Efficiency Management Science, Vol. 15 May 1969
   pp. 481-493

References Continued

8. Woolsey, R.E.D.
A Candle to Saint Jude, Or Four Real World Applications of Integer Programming
Interface, Vol. 2, No. 2, Feb 1972, pp. 20-27
The article gives potential users of integer programming some rather disconcerting advice to reconsider carefully before using any integer programming code.

Zero-One Linear Programming

See ref. 1, 4-7

Heuristic Zero-One Linear Programming

9. Zanakis, Stelios H.
Heuristic 0-1 Linear Programming:
An Experimental Comparison of Three Methods.
Managment Science, Vol. 24, No. 1, September 1977, pp. 91-104
This paper an excellent and detailed examination of the performance of three heuristic methods (Senju-Toyoda, Kochenberger, and Hillier) when applied to the 0-1 linear programming problem.

10. Toyoda, Yoshiaki
A Simplified Algorithm for Obtaining Approximate Solution to Zero-One Programming problems.
Management Science, Vol. 21, No. 12, August 1975, pp.1417-1427
Toyoda's original paper on his heuristic method. Contain a description of the method and experimental results.

11. Kochenberger, G.A.
A Heuristic for General Integer Programming
Decision Science, Vol. 5, No. 1, 1974, pp. 36-44
Kochenberger's heuristic method is described and examples given.